# CNN Model for Recognizing and Classifying Hand Gestures to Achieve Touchless User Interfaces

Ahmed Al-Sabounchi

May 17, 2019

# Definition

## Project Overview

Touchless interfaces have long been considered science fiction, only to be seen in futuristic sci-fi movies and video games, or part of the distant future. However, with the inception of modern Machine Learning techniques, many of these “distant future” concepts became a current day reality.

Companies such as Intel, Nvidia, and Microsoft have already began employing Neural Networks and Support Vector Machines to develop models that are able to recognize visual input in the form of gestures and translate it into computer output. This could open up the possibility of touchless human interfaces in the future, which would be very beneficial in a vast number of industries such as the medical, automotive, and entertainment industries, just to name a few [1].

## Problem Statement

In this project, the goal is to create a model that can identify 10 different hand gestures to a great degree of accuracy. This would help make a fast and accurate user interface that saves the user time and adds safety to some of their essential daily tasks, such as driving. In order to achieve this goal, an image dataset consisting of approximately 20,000 images of 10 different gestures taken by 10 individuals will be fed to a Convolutional Neural Network (CNN).

This CNN was constructed using Transfer Learning, where a pre-trained CNN was used to obtain the convolutional layers, followed by a Global Average Pooling (GAP) layer to flatten the data before passing it through a 10-node Dense layer with a SoftMax activation function to give 1 out of 10 outputs. The particular pre-trained architecture was chosen through a process of trial and error, where different architectures were evaluated based on their accuracies, where the model that yielded the highest accuracy on the training set was chosen.

## Metrics

To evaluate the model, a benchmark model was set up for comparison. A Confusion Matrix (CM) was constructed to obtain the Precision and Recall values for both models. With Precision defined as $Precision = 100 \times \frac{True\ Positives}{True\ Positives + False\ Positives}$, this was the metric that was used to quantify the accuracy of the proposed model and compare it to the benchmark. This model was chosen because of it's common use in classification problems.

## Analysis

### Data Exploration

The "Hand Gesture Recognition Database", provided by Leap Motion, was used to train the CNN. This dataset, which was obtained from Kaggle, contains around 20,000 images of 10 individuals (5 male, 5 female) making 10 distinct hand gestures [2]. The first image in this dataset is labelled as follows: frame_00_01_0001, where the 00 corresponds to the individual's identifier (0 to 9), the 01 corresponds to the hand gesture (1 to 10), and 0001 corresponds to the specific image of the gesture made by that individual (there are 200 images taken of the same gesture by each individual, to add for translational invariance).

For this project, the images were grouped based on the gesture. This means that pictures of different individuals performing the same action were grouped together. The model was fed 10 files corresponding to each gesture with 1,800 images each, adding up to 18,000 images (last 2,000 images were used as the testing set), and it classified the photos based on the file they were located in.

### Exploratory Visualization

Figure 1 displays the image input into the model. It can be observed that these images are input as 640x240 grayscale images before undergoing the required pre-processing. One noteworthy observation here is that the hand is not located in the same position in all of the images, which was something to keep in mind while modifying the images before passing them through the model.

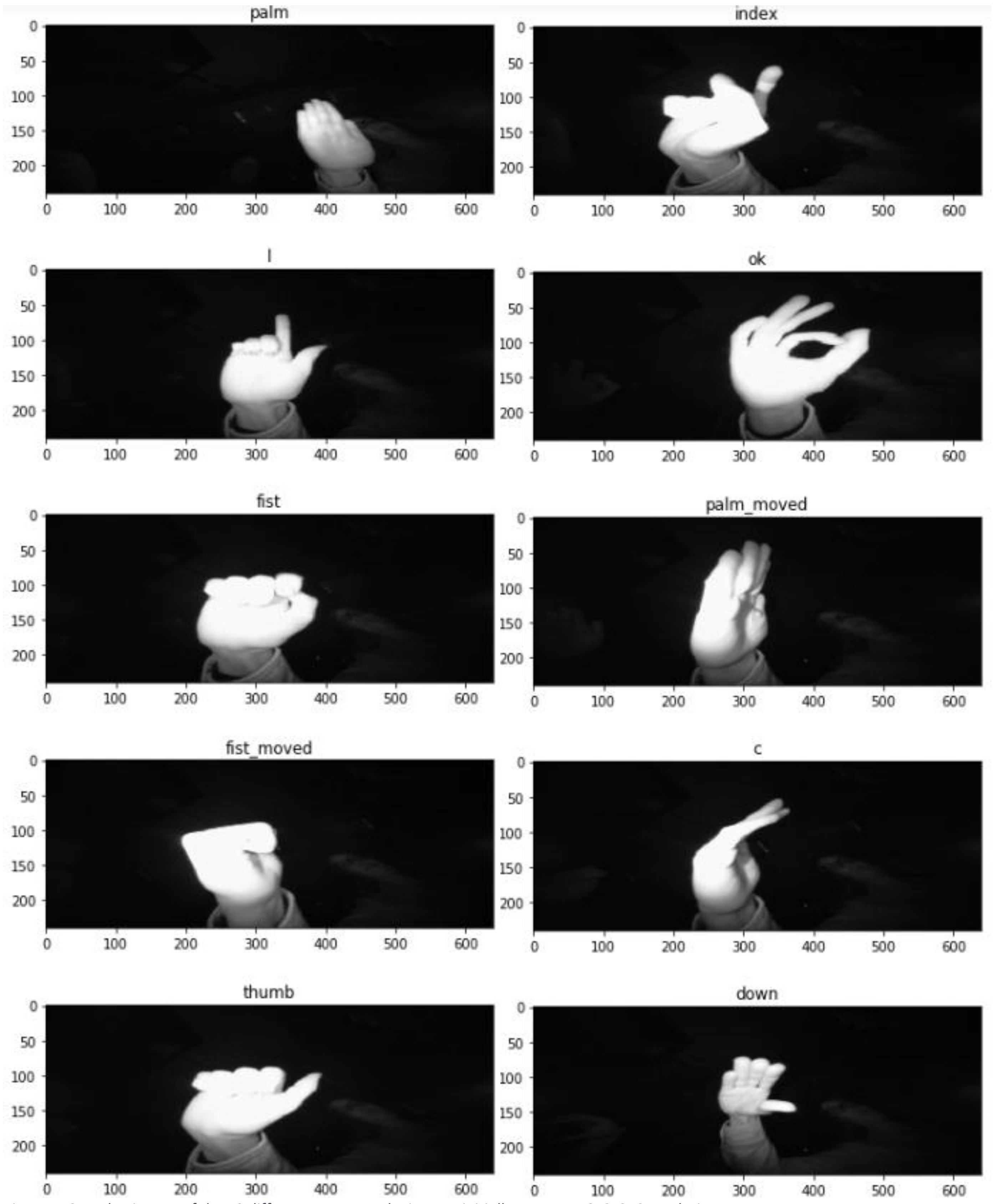

*Figure 1. Sample pictures of the 10 different gestures. The images initially possess a 640x240 resolution*

## Algorithms and Techniques

As mentioned earlier, Convolutional Neural Networks (CNNs) were used to solve this problem. CNNs are a class of Deep Neural Networks that use sparsely connected layers to avoid overfitting and to preserve the spatial information that is lost in Multi-Layer Perceptrons (MLPs) when the data is flattened. For this project, a CNN was first created from scratch to obtain a baseline score, then Transfer Learning was employed to obtain a more accurate model.

In Transfer Learning, highly accurate pre-trained models that were created to classify images in the ImageNet database were used to identify the more general features in the given dataset, before the images were flattened into vectors before passed through un-trained dense layers at the end to classify the more specific features. By doing this, the accuracy of the model is greatly increased, and the training time is reduced.

## Benchmark

First, a CNN was created from scratch without the help of pre-trained models to get a baseline accuracy of a simple CNN. Following that, the data was processed by 4 different pre-trained architectures (VGG16, ResNet50, Inception V3, and Xception) to determine the most appropriate architecture for the given data. Finally, the accuracy of the most appropriate model was evaluated against a Support Vector Machine (SVM) model that was created for the same dataset.

# Methodology

## Data Preprocessing

As was shown in Figure 1, the dataset consisted of 20,000 images with a resolution of 640x240. A reasonable assumption was made with regards to the color of the images, where the images were first converted into grayscale to cut down the input size down to one third, since the image color has no significant bearing on the classification result in this context.

After converting the images into grayscale, the image resolution was still too high to achieve a reasonable training time for the model, so the image resolution had to be decreased. Since the hand in each image was not located in the exact same position, cropping the image was not an option as it would risk losing some of the images. Therefore, the images were scaled down from 640x240 to 150x75, as shown in Figure 2. After observing the potential changes in the testing result in a smaller sub-sample, it was determined that the accuracy was not affected by this change, but the training time was reduced by roughly a magnitude of 10.

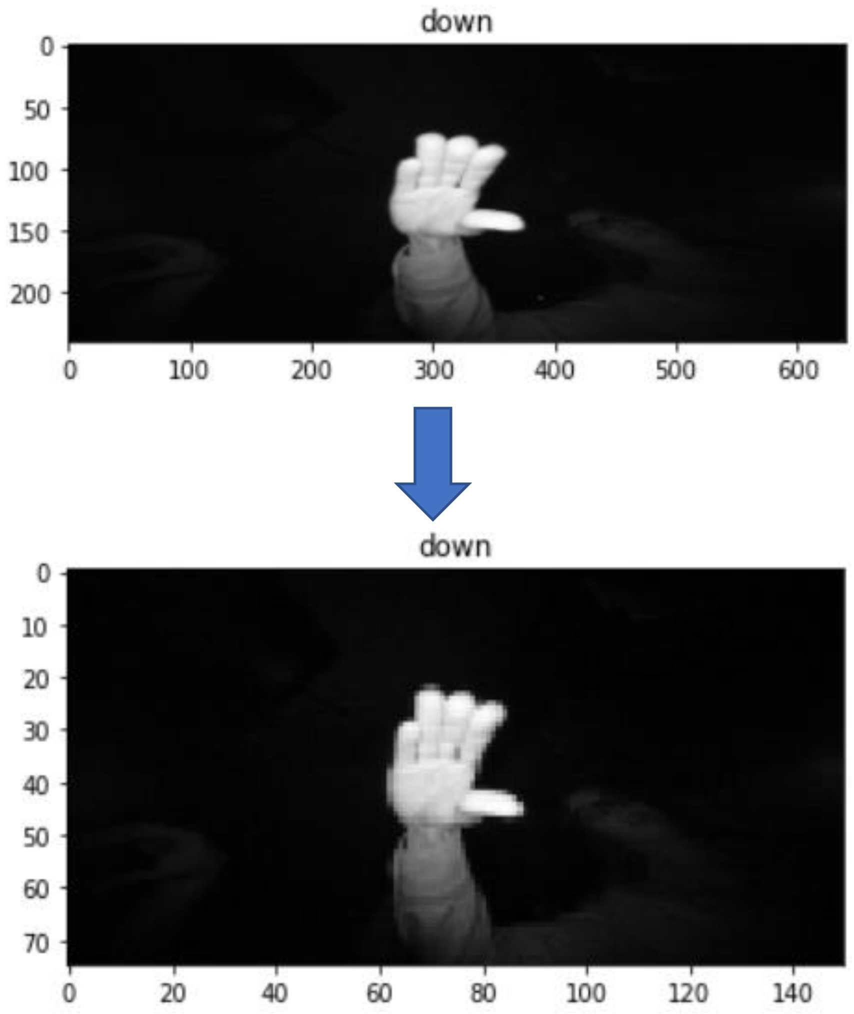

*Figure 2. Image resolution reduced from 640x240 to 150x75 to reduce computation time*

## Implementation

For the exact implementation of the algorithms to obtain the results discussed, please refer to the 'Model From Scratch.ipynb' and the 'Transfer Models.ipynb' Jupyter notebooks.

In making the CNN from scratch, the images were separated by gesture in 10 different files and placed in one directory. A list of all the gestures was made to convert the binary classifier for each image into a more meaningful string. The images were read in grayscale using cv2.imread then re-scaled to a smaller resolution before being paired with their corresponding labels to create the training set. The training set was shuffled using random.shuffle before being fed to the model.

The model constisted of 3 convolution layers with 64, 32, and 16 nodes, respectively, with ReLU activation functions and max pooling layers between every 2 convolution layers. A global average pooling layer was then used to flatten the data into a vector at the end before passing it into 2 dense layers to yield a 1 out of 10 classification result.

For the transfer learning models, bottleneck features needed to be obtained first. This was done using the model.predict_generator keras method to generate the bottleneck features from the given dataset. The models here consisted of the bottleneck features output, which were flattened into vectors before passing through a 512 node Dense layer and a final 10 node Dense layer with a softmax activation function to yield a classification result. This process was repeated for the 4 different architectures mentioned in the Algorithms and Techniques section to determine which model works best for this application.

Most of the complications in this process were encountered in the transfer learning section, due to the nature of the validation_split in the model.fit method, as it was discovered that it does not take a random validation sample, but rather it just splits the data from the end of the training set. Since the images were not initially shuffled, this led to 0% validation accuracy. Shuffling the training data before fitting the model solved this problem.

### Refinement

To maintain a controlled environment for testing, the same hyperparameters were used for all of the models, including the number of epochs, batch size, and image dimensions. After the first round, it was clear that the VGG16 model was the most suitable, so it was used to optimize the results. The number of layers in the dense layer, number of epochs, validation split, dropout, and type of flattening layer (Flatten vs. GlobalPoolingAverage2D) were manipulated to get the highest possible model accuracy, then all of these settings were applied to the other models. Manipulating these parameters resulted in a test accuracy increase from 82.21% to 89.90%.

## Results

### Model Evaluation and Validation

Out of the 20,000 images, 18,000 were used for testing an validation, leaving the last 2,000 images to be used as the testing test. All of the pictures in the testing set were taken by a different individual from the fist 18,000 images, in order to truly test the data with pictures that it has never seen before. After testing the different available architectures, the VGG16 model gave the best testing accuracy. By training the model for 20 epochs, it was able to reach 89.90% accuracy. This process was repeated 3 times, where the model achieved accuracies of 89.20%, 89.64%, and 89.90%, concluding that the model was able to consistently reach the same accuracy given constant training time.

After rating the different architectures' accuracies using the testing set and selecting the best architecture, that model was tested using random images from the internet to assess its effectiveness in classifying external images. Figure 3 shows some of the images that were passed through the model. Overall, 5 different images were tested and the model was able to correctly classify 4 of them, translating into an accuracy of 80%. Since this is not a binary classification problem and there are 10 different classes to classify, reaching a high accuracy was not going to be a simple task. In addition, all of the images passed had a very dark background with no edges, making it difficult to train the model to classify images with messier backgrounds. This makes the model good enough to classify images with clear backgrounds, but not very reliable in more realistic situations.

## Justification

An SVM model that was conducted on the same dataset by a user named Ninaand was selected as the benchmark model [4]. This model has an 88.75% accuracy, which is almost identical to the CNN's accuracy since the difference between the two models bears almost no statistical significance due to the variance in the CNN's accuracy results upon every run. This proves that the proposed model provides an adequate alternative to the existing benchmark, but it does not provide a better result. In the end, it becomes a matter of amount of time and computation power available. If these two things are readily available, perhaps a more complex CNN would be able to yield an even higher accuracy.

## Conclusion

### Free-Form Visualization

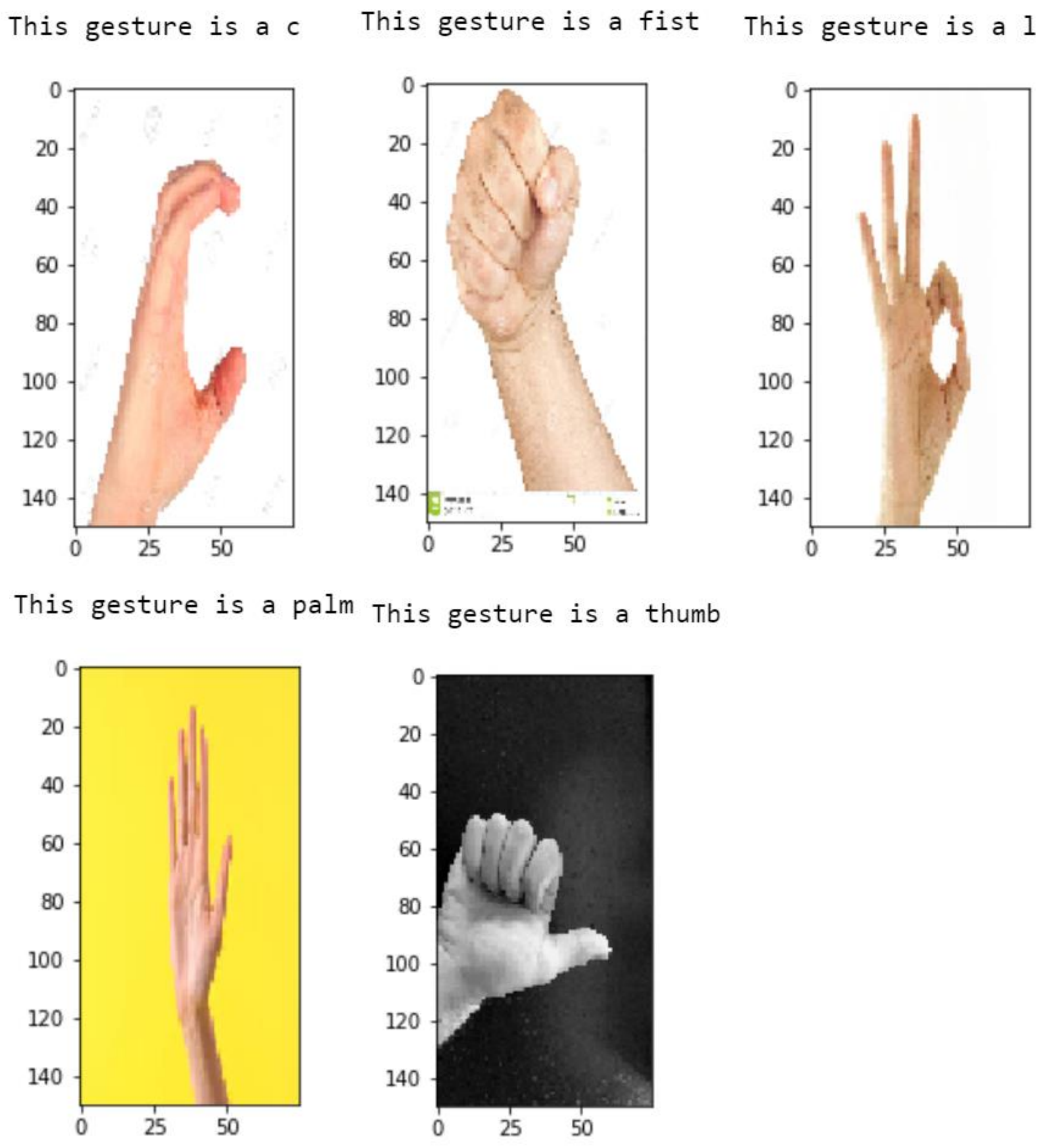

*Figure 3. Testing the model on 5 random images obtained from Google proved that the model is able to correctly carry our its intended function to a certain degree. However, it also proved that the model is less accurate than the suggested 89.90% reported accuracy.*

As mentioned in the Model Evaluation and Validation section, when tested with 5 random google images with clear backgrounds, the model was able to correctly predict 4 out of 5 of them. However, that number becomes much lower with more complex backgrounds, or if the same gestures were inverted. These issues can be addressed through image augmentation and the introduction of images with different backgrounds. However, this would require more effort in procuring these additional samples, and more computational power to actually compute all these images.

## Reflection

The problem statement was to construct a CNN capable of classifying 10 different hand gestures given 20,000 images. A CNN was constructed from scratch initially with 3 convolution layers and 2 dense layers to establish a baseline accuracy for the model to compare the pre-trained architectures to. This model yielded an accuracy of 72.90% when tested on the testing set.

Once the baseline has been established, different existing pre-trained models were tested on the dataset to get the best model. Out of the four architectures (VGG16, Inception V3, ResNet50, and Xception), VGG16 was determined to be the best model according to the accuracy reading. This was followed by parameter fine-tuning in an attempt to maximize the model's accuracy. This part was particularly time consuming due to the limited available computational power and the amount of time it took per run.

In the end, the proposed model was able to match existing architectures, but given more computational power, the model accuracy can be improved by augmenting the available data to represent a wider array of real-life scenarios.

## Improvement

To further increase this model's accuracy, there are a number of options that can be explored:

- Augment the data by altering the object's location, color, and other features to make the model more robust to handle real scenarios and less prone to overfitting
- Attempt to attain more image samples to give the model more training data also in an attempt to avoid overfitting
- Add more hidden layers to the model architecture to increase its accuracy
- Increase the number of epochs to give the model more time to train and ensure that the model's accuracy converges to the highest possible value

- The model had difficulty dealing with images with more complex backgrounds. Providing it with similar images in the training set would also make it more robust

## References

[1] P. Molchanov, S. Gupta, K. Kim and J. Kautz, "Hand gesture recognition with 3D convolutional neural networks", *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015. Available: 10.1109/cvprw.2015.7301342 [Accessed 8 May 2019].

[2] "Hand Gesture Recognition Database", *Kaggle.com*, 2019. [Online]. Available: https://www.kaggle.com/gti-upm/leapgestrecog. [Accessed: 08- May- 2019].
[3] D. Huang, W. Hu and S. Chang, "Gabor filter-based hand-pose angle estimation for hand gesture recognition under varying illumination", *Expert Systems with Applications*, vol. 38, no. 5, pp. 6031-6042, 2011. Available: 10.1016/j.eswa.2010.11.016 [Accessed 9 May 2019].
[4] “Ensemble approach for hand gesture recognition | Kaggle", *Kaggle.com*, 2019. [Online]. Available: https://www.kaggle.com/ninaad/ensemble-approach-for-hand-gesture-recognition#SVM. [Accessed: 16- May- 2019].